Network Working Group
Request for Comments: 1883
Category: Standards Track

S. Deering, Xerox PARC
R.  Hinden, Ipsilon Networks
December 1995

# Internet Protocol, Version 6 (IPv6) Specification

## Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements.  Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol.  Distribution of this memo is unlimited.

## Abstract

This document specifies version 6 of the Internet Protocol (IPv6), also sometimes referred to as IP Next Generation or IPng.

Table of Contents

## 1. Introduction

IP version 6 (IPv6) is a new version of the Internet Protocol, designed as a successor to IP version 4 (IPv4) [RFC-791]. The changes from IPv4 to IPv6 fall primarily into the following categories:

- Expanded Addressing Capabilities

  IPv6 increases the IP address size from 32 bits to 128 bits, to support more levels of addressing hierarchy, a much greater number of addressable nodes, and simpler auto-configuration of addresses. The scalability of multicast routing is improved by adding a "scope" field to multicast addresses. And a new type of address called an "anycast address" is defined, used to send a packet to any one of a group of nodes.

- Header Format Simplification

  Some IPv4 header fields have been dropped or made optional, to reduce the common-case processing cost of packet handling and to limit the bandwidth cost of the IPv6 header.

- Improved Support for Extensions and Options

  Changes in the way IP header options are encoded allows for more efficient forwarding, less stringent limits on the length of options, and greater flexibility for introducing new options in the future.

- Flow Labeling Capability

  A new capability is added to enable the labeling of packets belonging to particular traffic "flows" for which the sender requests special handling, such as non-default quality of service or "real-time" service.

- Authentication and Privacy Capabilities

  Extensions to support authentication, data integrity, and (optional) data confidentiality are specified for IPv6.

This document specifies the basic IPv6 header and the initially-defined IPv6 extension headers and options. It also discusses packet size issues, the semantics of flow labels and priority, and the effects of IPv6 on upper-layer protocols. The format and semantics of IPv6 addresses are specified separately in [RFC-1884]. The IPv6 version of ICMP, which all IPv6 implementations are required to include, is specified in [RFC-1885].

## 2. Terminology

node - a device that implements IPv6.

router - a node that forwards IPv6 packets not explicitly addressed to itself. [See Note below].

host - any node that is not a router. [See Note below].

upper layer - a protocol layer immediately above IPv6. Examples are transport protocols such as TCP and UDP, control protocols such as ICMP, routing protocols such as OSPF, and internet or lower-layer protocols being "tunneled" over (i.e., encapsulated in) IPv6 such as IPX, AppleTalk, or IPv6 itself.

link - a communication facility or medium over which nodes can communicate at the link layer, i.e., the layer immediately below IPv6. Examples are Ethernets (simple or bridged); PPP links; X.25, Frame Relay, or ATM networks; and internet (or higher) layer "tunnels", such as tunnels over IPv4 or IPv6 itself.

neighbors - nodes attached to the same link.

interface - a node’s attachment to a link.

address - an IPv6-layer identifier for an interface or a set of interfaces.

packet - an IPv6 header plus payload.

link MTU - the maximum transmission unit, i.e., maximum packet size in octets, that can be conveyed in one piece over a link.

path MTU - the minimum link MTU of all the links in a path between a source node and a destination node.

Note: it is possible, though unusual, for a device with multiple interfaces to be configured to forward non-self-destined packets arriving from some set (fewer than all) of its interfaces, and to discard non-self-destined packets arriving from its other interfaces. Such a device must obey the protocol requirements for routers when receiving packets from, and interacting with neighbors over, the former (forwarding) interfaces. It must obey the protocol requirements for hosts when receiving packets from, and interacting with neighbors over, the latter (non-forwarding) interfaces.

## 3. IPv6 Header Format

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Prio. |                   Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version — 4-bit Internet Protocol version number = 6.

Prio. — 4-bit priority value. See section 7.

Flow Label — 24-bit flow label. See section 6.

Payload Length — 16-bit unsigned integer. Length of payload, i.e., the rest of the packet following the IPv6 header, in octets. If zero, indicates that the payload length is carried in a Jumbo Payload hop-by-hop option.

Next Header — 8-bit selector. Identifies the type of header immediately following the IPv6 header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].

Hop Limit — 8-bit unsigned integer. Decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero.

Source Address — 128-bit address of the originator of the packet. See [RFC-1884].

Destination Address 128-bit address of the intended recipient of the packet (possibly not the ultimate recipient, if a Routing header is present). See [RFC-1884] and section 4.4.

## 4. IPv6 Extension Headers

In IPv6, optional internet-layer information is encoded in separate headers that may be placed between the IPv6 header and the upper-layer header in a packet. There are a small number of such extension headers, each identified by a distinct Next Header value. As illustrated in these examples, an IPv6 packet may carry zero, one, or more extension headers, each identified by the Next Header field of the preceding header:

```
+---------------+------------------------
|  IPv6 header  | TCP header + data
|               |
| Next Header = |
|      TCP      |
+---------------+------------------------


+---------------+----------------+------------------------
|  IPv6 header  | Routing header | TCP header + data
|               |                |
| Next Header = |  Next Header = |
|    Routing    |      TCP       |
+---------------+----------------+------------------------


+---------------+----------------+-----------------+-----------------
|  IPv6 header  | Routing header | Fragment header | fragment of TCP
|               |                |                 |  header + data
| Next Header = |  Next Header = |  Next Header =  |
|    Routing    |    Fragment    |       TCP       |
+---------------+----------------+-----------------+-----------------
```

With one exception, extension headers are not examined or processed by any node along a packet's delivery path, until the packet reaches the node (or each of the set of nodes, in the case of multicast) identified in the Destination Address field of the IPv6 header. There, normal demultiplexing on the Next Header field of the IPv6 header invokes the module to process the first extension header, or the upper-layer header if no extension header is present. The contents and semantics of each extension header determine whether or

not to proceed to the next header. Therefore, extension headers must be processed strictly in the order they appear in the packet; a receiver must not, for example, scan through a packet looking for a particular kind of extension header and process that header prior to processing all preceding ones.

The exception referred to in the preceding paragraph is the Hop-by-Hop Options header, which carries information that must be examined and processed by every node along a packet's delivery path, including the source and destination nodes. The Hop-by-Hop Options header, when present, must immediately follow the IPv6 header. Its presence is indicated by the value zero in the Next Header field of the IPv6 header.

If, as a result of processing a header, a node is required to proceed to the next header but the Next Header value in the current header is unrecognized by the node, it should discard the packet and send an ICMP Parameter Problem message to the source of the packet, with an ICMP Code value of 2 ("unrecognized Next Header type encountered") and the ICMP Pointer field containing the offset of the unrecognized value within the original packet. The same action should be taken if a node encounters a Next Header value of zero in any header other than an IPv6 header.

Each extension header is an integer multiple of 8 octets long, in order to retain 8-octet alignment for subsequent headers. Multi-octet fields within each extension header are aligned on their natural boundaries, i.e., fields of width n octets are placed at an integer multiple of n octets from the start of the header, for n = 1, 2, 4, or 8.

A full implementation of IPv6 includes implementation of the following extension headers:

Hop-by-Hop Options
Routing (Type 0)
Fragment
Destination Options
Authentication
Encapsulating Security Payload

The first four are specified in this document; the last two are specified in [RFC-1826] and [RFC-1827], respectively.

## 4.1 Extension Header Order

When more than one extension header is used in the same packet, it is recommended that those headers appear in the following order:

IPv6 header
Hop-by-Hop Options header
Destination Options header (note 1)
Routing header
Fragment header
Authentication header (note 2)
Encapsulating Security Payload header (note 2)
Destination Options header (note 3)
upper-layer header

note 1: for options to be processed by the first destination that appears in the IPv6 Destination Address field plus subsequent destinations listed in the Routing header.

note 2: additional recommendations regarding the relative order of the Authentication and Encapsulating Security Payload headers are given in [RFC-1827].

note 3: for options to be processed only by the final destination of the packet.

Each extension header should occur at most once, except for the Destination Options header which should occur at most twice (once before a Routing header and once before the upper-layer header).

If the upper-layer header is another IPv6 header (in the case of IPv6 being tunneled over or encapsulated in IPv6), it may be followed by its own extensions headers, which are separately subject to the same ordering recommendations.

If and when other extension headers are defined, their ordering constraints relative to the above listed headers must be specified.

IPv6 nodes must accept and attempt to process extension headers in any order and occurring any number of times in the same packet, except for the Hop-by-Hop Options header which is restricted to appear immediately after an IPv6 header only. Nonetheless, it is strongly advised that sources of IPv6 packets adhere to the above recommended order until and unless subsequent specifications revise that recommendation.

### 4.2 Options

Two of the currently-defined extension headers -- the Hop-by-Hop Options header and the Destination Options header -- carry a variable number of type-length-value (TLV) encoded "options", of the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
|  Option Type  |  Opt Data Len |  Option Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
```

Option Type — 8-bit identifier of the type of option.

Opt Data Len — 8-bit unsigned integer. Length of the Option Data field of this option, in octets.

Option Data — Variable-length field. Option-Type-specific data.

The sequence of options within a header must be processed strictly in the order they appear in the header; a receiver must not, for example, scan through the header looking for a particular kind of option and process that option prior to processing all preceding ones.

The Option Type identifiers are internally encoded such that their highest-order two bits specify the action that must be taken if the processing IPv6 node does not recognize the Option Type:

00 - skip over this option and continue processing the header.

01 - discard the packet.

10 - discard the packet and, regardless of whether or not the packets's Destination Address was a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type.

11 - discard the packet and, only if the packet's Destination Address was not a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type.

The third-highest-order bit of the Option Type specifies whether or not the Option Data of that option can change en-route to the packet's final destination. When an Authentication header is present in the packet, for any option whose data may change en-route, its entire Option Data field must be treated as zero-valued octets when computing or verifying the packet's authenticating value.

0 - Option Data does not change en-route

1 - Option Data may change en-route

Individual options may have specific alignment requirements, to ensure that multi-octet values within Option Data fields fall on natural boundaries. The alignment requirement of an option is specified using the notation xn+y, meaning the Option Type must appear at an integer multiple of x octets from the start of the header, plus y octets. For example:

```
2n    means any 2-octet offset from the start of the header.
8n+2  means any 8-octet offset from the start of the header,
      plus 2 octets.
```

There are two padding options which are used when necessary to align subsequent options and to pad out the containing header to a multiple of 8 octets in length. These padding options must be recognized by all IPv6 implementations:

Pad1 option (alignment requirement: none)

```
+-+-+-+-+-+-+-+-+
|       0       |
+-+-+-+-+-+-+-+-+
```

NOTE! the format of the Pad1 option is a special case -- it does not have length and value fields.

The Pad1 option is used to insert one octet of padding into the Options area of a header. If more than one octet of padding is required, the PadN option, described next, should be used, rather than multiple Pad1 options.

PadN option (alignment requirement: none)

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
|       1       |  Opt Data Len |  Option Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
```

The PadN option is used to insert two or more octets of padding into the Options area of a header. For N octets of padding, the Opt Data Len field contains the value N-2, and the Option Data consists of N-2 zero-valued octets.

Appendix A contains formatting guidelines for designing new options.

## 4.3  Hop-by-Hop Options Header

The Hop-by-Hop Options header is used to carry optional information that must be examined by every node along a packet's delivery path. The Hop-by-Hop Options header is identified by a Next Header value of 0 in the IPv6 header, and has the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header          8-bit selector.  Identifies the type of header immediately following the Hop-by-Hop Options header.  Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].

Hdr Ext Len          8-bit unsigned integer.  Length of the Hop-by-Hop Options header in 8-octet units, not including the first 8 octets.

Options              Variable-length field, of length such that the complete Hop-by-Hop Options header is an integer multiple of 8 octets long.  Contains one or more TLV-encoded options, as described in section 4.2.

In addition to the Pad1 and PadN options specified in section 4.2, the following hop-by-hop option is defined:

Jumbo Payload option  (alignment requirement: 4n + 2)

```
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |      194      |Opt Data Len=4 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Jumbo Payload Length                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Jumbo Payload option is used to send IPv6 packets with payloads longer than 65,535 octets.  The Jumbo Payload Length is the length of the packet in octets, excluding the IPv6 header but including the Hop-by-Hop Options header; it must be greater than 65,535.  If a packet is received with a Jumbo Payload option containing a Jumbo Payload Length less than or equal to 65,535,

an ICMP Parameter Problem message, Code 0, should be sent to the packet's source, pointing to the high-order octet of the invalid Jumbo Payload Length field.

The Payload Length field in the IPv6 header must be set to zero in every packet that carries the Jumbo Payload option. If a packet is received with a valid Jumbo Payload option present and a non-zero IPv6 Payload Length field, an ICMP Parameter Problem message, Code 0, should be sent to the packet's source, pointing to the Option Type field of the Jumbo Payload option.

The Jumbo Payload option must not be used in a packet that carries a Fragment header. If a Fragment header is encountered in a packet that contains a valid Jumbo Payload option, an ICMP Parameter Problem message, Code 0, should be sent to the packet's source, pointing to the first octet of the Fragment header.

An implementation that does not support the Jumbo Payload option cannot have interfaces to links whose link MTU is greater than 65,575 (40 octets of IPv6 header plus 65,535 octets of payload).

## 4.4 Routing Header

The Routing header is used by an IPv6 source to list one or more intermediate nodes to be "visited" on the way to a packet's destination. This function is very similar to IPv4's Source Route options. The Routing header is identified by a Next Header value of 43 in the immediately preceding header, and has the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |  Routing Type | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                       type-specific data                      .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

| | |
|---|---|
| Next Header | 8-bit selector. Identifies the type of header immediately following the Routing header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.]. |
| Hdr Ext Len | 8-bit unsigned integer. Length of the Routing header in 8-octet units, not including the first 8 octets. |
| Routing Type | 8-bit identifier of a particular Routing header variant. |
| Segments Left | 8-bit unsigned integer. Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination. |
| type-specific data | Variable-length field, of format determined by the Routing Type, and of length such that the complete Routing header is an integer multiple of 8 octets long. |

If, while processing a received packet, a node encounters a Routing header with an unrecognized Routing Type value, the required behavior of the node depends on the value of the Segments Left field, as follows:

If Segments Left is zero, the node must ignore the Routing header and proceed to process the next header in the packet, whose type is identified by the Next Header field in the Routing header.

If Segments Left is non-zero, the node must discard the packet and send an ICMP Parameter Problem, Code 0, message to the packet's Source Address, pointing to the unrecognized Routing Type.

The Type 0 Routing header has the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  | Routing Type=0| Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |             Strict/Loose Bit Map              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                           Address[1]                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                           Address[2]                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                               .                               .
.                               .                               .
.                               .                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                           Address[n]                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header — 8-bit selector. Identifies the type of header immediately following the Routing header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].

Hdr Ext Len — 8-bit unsigned integer. Length of the Routing header in 8-octet units, not including the first 8 octets. For the Type 0 Routing header, Hdr Ext Len is equal to two times the number of addresses in the header, and must be an even number less than or equal to 46.

Routing Type — 0.

Segments Left — 8-bit unsigned integer. Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination. Maximum legal value = 23.

Reserved — 8-bit reserved field. Initialized to zero for transmission; ignored on reception.

Strict/Loose Bit Map — 24-bit bit-map, numbered 0 to 23, left-to-right. Indicates, for each segment of the route, whether or not the next destination address must be a neighbor of the preceding address: 1 means strict (must be a neighbor), 0 means loose (need not be a neighbor).

Address[1..n] — Vector of 128-bit addresses, numbered 1 to n.

Multicast addresses must not appear in a Routing header of Type 0, or in the IPv6 Destination Address field of a packet carrying a Routing header of Type 0.

If bit number 0 of the Strict/Loose Bit Map has value 1, the Destination Address field of the IPv6 header in the original packet must identify a neighbor of the originating node. If bit number 0 has value 0, the originator may use any legal, non-multicast address as the initial Destination Address.

Bits numbered greater than n, where n is the number of addresses in the Routing header, must be set to 0 by the originator and ignored by receivers.

A Routing header is not examined or processed until it reaches the node identified in the Destination Address field of the IPv6 header. In that node, dispatching on the Next Header field of the immediately preceding header causes the Routing header module to be invoked, which, in the case of Routing Type 0, performs the following algorithm:

```
   if Segments Left = 0 {
      proceed to process the next header in the packet, whose type is
      identified by the Next Header field in the Routing header
   }
   else if Hdr Ext Len is odd or greater than 46 {
         send an ICMP Parameter Problem, Code 0, message to the Source
         Address, pointing to the Hdr Ext Len field, and discard the
         packet
   }
   else {
      compute n, the number of addresses in the Routing header, by
      dividing Hdr Ext Len by 2

      if Segments Left is greater than n {
         send an ICMP Parameter Problem, Code 0, message to the Source
         Address, pointing to the Segments Left field, and discard the
         packet
      }
      else {
         decrement Segments Left by 1;
         compute i, the index of the next address to be visited in
         the address vector, by subtracting Segments Left from n

         if Address [i] or the IPv6 Destination Address is multicast {
            discard the packet
         }
         else {
            swap the IPv6 Destination Address and Address[i]

            if bit i of the Strict/Loose Bit map has value 1 and the
            new Destination Address is not the address of a neighbor
            of this node {
               send an ICMP Destination Unreachable -- Not a Neighbor
               message to the Source Address and discard the packet
            }
            else if the IPv6 Hop Limit is less than or equal to 1 {
               send an ICMP Time Exceeded -- Hop Limit Exceeded in
               Transit message to the Source Address and discard the
               packet
            }
            else {
               decrement the Hop Limit by 1

               resubmit the packet to the IPv6 module for transmission
               to the new destination
            }
         }
      }
   }
```

As an example of the effects of the above algorithm, consider the case of a source node S sending a packet to destination node D, using a Routing header to cause the packet to be routed via intermediate nodes I1, I2, and I3. The values of the relevant IPv6 header and Routing header fields on each segment of the delivery path would be as follows:

As the packet travels from S to I1:

```
     Source Address = S                  Hdr Ext Len = 6
     Destination Address = I1            Segments Left = 3
                                         Address[1] = I2
     (if bit 0 of the Bit Map is 1,      Address[2] = I3
      S and I1 must be neighbors;        Address[3] = D
      this is checked by S)
```

As the packet travels from I1 to I2:

```
     Source Address = S                  Hdr Ext Len = 6
     Destination Address = I2            Segments Left = 2
                                         Address[1] = I1
     (if bit 1 of the Bit Map is 1,      Address[2] = I3
      I1 and I2 must be neighbors;       Address[3] = D
      this is checked by I1)
```

As the packet travels from I2 to I3:

```
     Source Address = S                  Hdr Ext Len = 6
     Destination Address = I3            Segments Left = 1
                                         Address[1] = I1
     (if bit 2 of the Bit Map is 1,      Address[2] = I2
      I2 and I3 must be neighbors;       Address[3] = D
      this is checked by I2)
```

As the packet travels from I3 to D:

```
     Source Address = S                  Hdr Ext Len = 6
     Destination Address = D             Segments Left = 0
                                         Address[1] = I1
     (if bit 3 of the Bit Map is 1,      Address[2] = I2
      I3 and D must be neighbors;        Address[3] = I3
      this is checked by I3)
```

## 4.5 Fragment Header

The Fragment header is used by an IPv6 source to send packets larger than would fit in the path MTU to their destinations. (Note: unlike IPv4, fragmentation in IPv6 is performed only by source nodes, not by routers along a packet's delivery path -- see section 5.) The Fragment header is identified by a Next Header value of 44 in the immediately preceding header, and has the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |   Reserved    |      Fragment Offset    |Res|M|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Identification                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header — 8-bit selector. Identifies the initial header type of the Fragmentable Part of the original packet (defined below). Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].

Reserved — 8-bit reserved field. Initialized to zero for transmission; ignored on reception.

Fragment Offset — 13-bit unsigned integer. The offset, in 8-octet units, of the data following this header, relative to the start of the Fragmentable Part of the original packet.

Res — 2-bit reserved field. Initialized to zero for transmission; ignored on reception.

M flag — 1 = more fragments; 0 = last fragment.

Identification — 32 bits. See description below.

In order to send a packet that is too large to fit in the MTU of the path to its destination, a source node may divide the packet into fragments and send each fragment as a separate packet, to be reassembled at the receiver.

For every packet that is to be fragmented, the source node generates an Identification value. The Identification must be different than that of any other fragmented packet sent recently* with the same Source Address and Destination Address. If a Routing header is present, the Destination Address of concern is that of the final destination.

* "recently" means within the maximum likely lifetime of a packet, including transit time from source to destination and time spent

awaiting reassembly with other fragments of the same packet. However, it is not required that a source node know the maximum packet lifetime. Rather, it is assumed that the requirement can be met by maintaining the Identification value as a simple, 32-bit, "wrap-around" counter, incremented each time a packet must be fragmented. It is an implementation choice whether to maintain a single counter for the node or multiple counters, e.g., one for each of the node's possible source addresses, or one for each active (source address, destination address) combination.

The initial, large, unfragmented packet is referred to as the "original packet", and it is considered to consist of two parts, as illustrated:

original packet:

```
+------------------+----------------------//-----------------------+
|  Unfragmentable  |                 Fragmentable                  |
|       Part       |                     Part                      |
+------------------+----------------------//-----------------------+
```

The Unfragmentable Part consists of the IPv6 header plus any extension headers that must be processed by nodes en route to the destination, that is, all headers up to and including the Routing header if present, else the Hop-by-Hop Options header if present, else no extension headers.

The Fragmentable Part consists of the rest of the packet, that is, any extension headers that need be processed only by the final destination node(s), plus the upper-layer header and data.

The Fragmentable Part of the original packet is divided into fragments, each, except possibly the last ("rightmost") one, being an integer multiple of 8 octets long. The fragments are transmitted in separate "fragment packets" as illustrated:

original packet:

```
+------------------+--------------+--------------+--//--+----------+
|  Unfragmentable  |    first     |    second    |      |   last   |
|       Part       |   fragment   |   fragment   | .... | fragment |
+------------------+--------------+--------------+--//--+----------+
```

fragment packets:

```
+------------------+--------+--------------+
|  Unfragmentable  |Fragment|    first     |
|       Part       | Header |   fragment   |
+------------------+--------+--------------+

+------------------+--------+--------------+
|  Unfragmentable  |Fragment|    second    |
|       Part       | Header |   fragment   |
+------------------+--------+--------------+
                      o
                      o
                      o
+------------------+--------+----------+
|  Unfragmentable  |Fragment|   last   |
|       Part       | Header | fragment |
+------------------+--------+----------+
```

Each fragment packet is composed of:

(1) The Unfragmentable Part of the original packet, with the Payload Length of the original IPv6 header changed to contain the length of this fragment packet only (excluding the length of the IPv6 header itself), and the Next Header field of the last header of the Unfragmentable Part changed to 44.

(2) A Fragment header containing:

The Next Header value that identifies the first header of the Fragmentable Part of the original packet.

A Fragment Offset containing the offset of the fragment, in 8-octet units, relative to the start of the Fragmentable Part of the original packet. The Fragment Offset of the first ("leftmost") fragment is 0.

An M flag value of 0 if the fragment is the last ("rightmost") one, else an M flag value of 1.

The Identification value generated for the original packet.

(3) The fragment itself.

The lengths of the fragments must be chosen such that the resulting fragment packets fit within the MTU of the path to the packets' destination(s).

At the destination, fragment packets are reassembled into their original, unfragmented form, as illustrated:

reassembled original packet:

```
+------------------+---------------------//------------------------+
|  Unfragmentable  |                 Fragmentable                  |
|       Part       |                     Part                      |
+------------------+---------------------//------------------------+
```

The following rules govern reassembly:

An original packet is reassembled only from fragment packets that have the same Source Address, Destination Address, and Fragment Identification.

The Unfragmentable Part of the reassembled packet consists of all headers up to, but not including, the Fragment header of the first fragment packet (that is, the packet whose Fragment Offset is zero), with the following two changes:

The Next Header field of the last header of the Unfragmentable Part is obtained from the Next Header field of the first fragment's Fragment header.

The Payload Length of the reassembled packet is computed from the length of the Unfragmentable Part and the length and offset of the last fragment. For example, a formula for computing the Payload Length of the reassembled original packet is:

```
PL.orig = PL.first - FL.first - 8 + (8 * FO.last) + FL.last

where
PL.orig  = Payload Length field of reassembled packet.
PL.first = Payload Length field of first fragment packet.
FL.first = length of fragment following Fragment header of
           first fragment packet.
FO.last  = Fragment Offset field of Fragment header of
           last fragment packet.
FL.last  = length of fragment following Fragment header of
           last fragment packet.
```

The Fragmentable Part of the reassembled packet is constructed from the fragments following the Fragment headers in each of the fragment packets. The length of each fragment is computed by subtracting from the packet's Payload Length the length of the headers between the IPv6 header and fragment itself; its relative position in Fragmentable Part is computed from its Fragment Offset value.

The Fragment header is not present in the final, reassembled packet.

The following error conditions may arise when reassembling fragmented packets:

If insufficient fragments are received to complete reassembly of a packet within 60 seconds of the reception of the first-arriving fragment of that packet, reassembly of that packet must be abandoned and all the fragments that have been received for that packet must be discarded. If the first fragment (i.e., the one with a Fragment Offset of zero) has been received, an ICMP Time Exceeded -- Fragment Reassembly Time Exceeded message should be sent to the source of that fragment.

If the length of a fragment, as derived from the fragment packet's Payload Length field, is not a multiple of 8 octets and the M flag of that fragment is 1, then that fragment must be discarded and an ICMP Parameter Problem, Code 0, message should be sent to the source of the fragment, pointing to the Payload Length field of the fragment packet.

If the length and offset of a fragment are such that the Payload Length of the packet reassembled from that fragment would exceed 65,535 octets, then that fragment must be discarded and an ICMP Parameter Problem, Code 0, message should be sent to the source of the fragment, pointing to the Fragment Offset field of the fragment packet.

The following conditions are not expected to occur, but are not considered errors if they do:

The number and content of the headers preceding the Fragment header of different fragments of the same original packet may differ. Whatever headers are present, preceding the Fragment header in each fragment packet, are processed when the packets arrive, prior to queueing the fragments for reassembly. Only those headers in the Offset zero fragment packet are retained in the reassembled packet.

The Next Header values in the Fragment headers of different fragments of the same original packet may differ. Only the value from the Offset zero fragment packet is used for reassembly.

## 4.6 Destination Options Header

The Destination Options header is used to carry optional information that need be examined only by a packet's destination node(s). The Destination Options header is identified by a Next Header value of 60 in the immediately preceding header, and has the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header — 8-bit selector. Identifies the type of header immediately following the Destination Options header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].

Hdr Ext Len — 8-bit unsigned integer. Length of the Destination Options header in 8-octet units, not including the first 8 octets.

Options — Variable-length field, of length such that the complete Destination Options header is an integer multiple of 8 octets long. Contains one or more TLV-encoded options, as described in section 4.2.

The only destination options defined in this document are the Pad1 and PadN options specified in section 4.2.

Note that there are two possible ways to encode optional destination information in an IPv6 packet: either as an option in the Destination Options header, or as a separate extension header. The Fragment header and the Authentication header are examples of the latter approach. Which approach can be used depends on what action is desired of a destination node that does not understand the optional information:

- o if the desired action is for the destination node to discard the packet and, only if the packet's Destination Address is not a multicast address, send an ICMP Unrecognized Type message to the packet's Source Address, then the information may be encoded either as a separate header or as an option in the

Destination Options header whose Option Type has the value 11 in its highest-order two bits. The choice may depend on such factors as which takes fewer octets, or which yields better alignment or more efficient parsing.

- o if any other action is desired, the information must be encoded as an option in the Destination Options header whose Option Type has the value 00, 01, or 10 in its highest-order two bits, specifying the desired action (see section 4.2).

## 4.7 No Next Header

The value 59 in the Next Header field of an IPv6 header or any extension header indicates that there is nothing following that header. If the Payload Length field of the IPv6 header indicates the presence of octets past the end of a header whose Next Header field contains 59, those octets must be ignored, and passed on unchanged if the packet is forwarded.

## 5. Packet Size Issues

IPv6 requires that every link in the internet have an MTU of 576 octets or greater. On any link that cannot convey a 576-octet packet in one piece, link-specific fragmentation and reassembly must be provided at a layer below IPv6.

From each link to which a node is directly attached, the node must be able to accept packets as large as that link's MTU. Links that have a configurable MTU (for example, PPP links [RFC-1661]) must be configured to have an MTU of at least 576 octets; it is recommended that a larger MTU be configured, to accommodate possible encapsulations (i.e., tunneling) without incurring fragmentation.

It is strongly recommended that IPv6 nodes implement Path MTU Discovery [RFC-1191], in order to discover and take advantage of paths with MTU greater than 576 octets. However, a minimal IPv6 implementation (e.g., in a boot ROM) may simply restrict itself to sending packets no larger than 576 octets, and omit implementation of Path MTU Discovery.

In order to send a packet larger than a path's MTU, a node may use the IPv6 Fragment header to fragment the packet at the source and have it reassembled at the destination(s). However, the use of such fragmentation is discouraged in any application that is able to adjust its packets to fit the measured path MTU (i.e., down to 576 octets).

A node must be able to accept a fragmented packet that, after reassembly, is as large as 1500 octets, including the IPv6 header. A node is permitted to accept fragmented packets that reassemble to more than 1500 octets. However, a node must not send fragments that reassemble to a size greater than 1500 octets unless it has explicit knowledge that the destination(s) can reassemble a packet of that size.

In response to an IPv6 packet that is sent to an IPv4 destination (i.e., a packet that undergoes translation from IPv6 to IPv4), the originating IPv6 node may receive an ICMP Packet Too Big message reporting a Next-Hop MTU less than 576. In that case, the IPv6 node is not required to reduce the size of subsequent packets to less than 576, but must include a Fragment header in those packets so that the IPv6-to-IPv4 translating router can obtain a suitable Identification value to use in resulting IPv4 fragments. Note that this means the payload may have to be reduced to 528 octets (576 minus 40 for the IPv6 header and 8 for the Fragment header), and smaller still if additional extension headers are used.

Note: Path MTU Discovery must be performed even in cases where a host "thinks" a destination is attached to the same link as itself.

Note: Unlike IPv4, it is unnecessary in IPv6 to set a "Don't Fragment" flag in the packet header in order to perform Path MTU Discovery; that is an implicit attribute of every IPv6 packet. Also, those parts of the RFC-1191 procedures that involve use of a table of MTU "plateaus" do not apply to IPv6, because the IPv6 version of the "Datagram Too Big" message always identifies the exact MTU to be used.

## 6. Flow Labels

The 24-bit Flow Label field in the IPv6 header may be used by a source to label those packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service. This aspect of IPv6 is, at the time of writing, still experimental and subject to change as the requirements for flow support in the Internet become clearer. Hosts or routers that do not support the functions of the Flow Label field are required to set the field to zero when originating a packet, pass the field on unchanged when forwarding a packet, and ignore the field when receiving a packet.

A flow is a sequence of packets sent from a particular source to a particular (unicast or multicast) destination for which the source desires special handling by the intervening routers. The nature of that special handling might be conveyed to the routers by a control protocol, such as a resource reservation protocol, or by information within the flow's packets themselves, e.g., in a hop-by-hop option. The details of such control protocols or options are beyond the scope of this document.

There may be multiple active flows from a source to a destination, as well as traffic that is not associated with any flow. A flow is uniquely identified by the combination of a source address and a non-zero flow label. Packets that do not belong to a flow carry a flow label of zero.

A flow label is assigned to a flow by the flow's source node. New flow labels must be chosen (pseudo-)randomly and uniformly from the range 1 to FFFFFF hex. The purpose of the random allocation is to make any set of bits within the Flow Label field suitable for use as a hash key by routers, for looking up the state associated with the flow.

All packets belonging to the same flow must be sent with the same source address, destination address, priority, and flow label. If any of those packets includes a Hop-by-Hop Options header, then they all must be originated with the same Hop-by-Hop Options header contents (excluding the Next Header field of the Hop-by-Hop Options header). If any of those packets includes a Routing header, then they all must be originated with the same contents in all extension headers up to and including the Routing header (excluding the Next Header field in the Routing header). The routers or destinations are permitted, but not required, to verify that these conditions are satisfied. If a violation is detected, it should be reported to the source by an ICMP Parameter Problem message, Code 0, pointing to the high-order octet of the Flow Label field (i.e., offset 1 within the IPv6 packet).

Routers are free to "opportunistically" set up flow-handling state for any flow, even when no explicit flow establishment information has been provided to them via a control protocol, a hop-by-hop option, or other means. For example, upon receiving a packet from a particular source with an unknown, non-zero flow label, a router may process its IPv6 header and any necessary extension headers as if the flow label were zero. That processing would include determining the next-hop interface, and possibly other actions, such as updating a hop-by-hop option, advancing the pointer and addresses in a Routing header, or deciding on how to queue the packet based on its Priority field. The router may then choose to "remember" the results of those processing steps and cache that information, using the source address plus the flow label as the cache key. Subsequent packets with the same source address and flow label may then be handled by referring to the cached information rather than examining all those fields that, according to the requirements of the previous paragraph, can be assumed unchanged from the first packet seen in the flow.

Cached flow-handling state that is set up opportunistically, as discussed in the preceding paragraph, must be discarded no more than 6 seconds after it is established, regardless of whether or not packets of the same flow continue to arrive. If another packet with the same source address and flow label arrives after the cached state has been discarded, the packet undergoes full, normal processing (as if its flow label were zero), which may result in the re-creation of cached flow state for that flow.

The lifetime of flow-handling state that is set up explicitly, for example by a control protocol or a hop-by-hop option, must be specified as part of the specification of the explicit set-up mechanism; it may exceed 6 seconds.

A source must not re-use a flow label for a new flow within the lifetime of any flow-handling state that might have been established for the prior use of that flow label. Since flow-handling state with a lifetime of 6 seconds may be established opportunistically for any flow, the minimum interval between the last packet of one flow and the first packet of a new flow using the same flow label is 6 seconds. Flow labels used for explicitly set-up flows with longer flow-state lifetimes must remain unused for those longer lifetimes before being re-used for new flows.

When a node stops and restarts (e.g., as a result of a "crash"), it must be careful not to use a flow label that it might have used for an earlier flow whose lifetime may not have expired yet. This may be accomplished by recording flow label usage on stable storage so that it can be remembered across crashes, or by refraining from using any flow labels until the maximum lifetime of any possible previously established flows has expired (at least 6 seconds; more if explicit

flow set-up mechanisms with longer lifetimes might have been used). If the minimum time for rebooting the node is known (often more than 6 seconds), that time can be deducted from the necessary waiting period before starting to allocate flow labels.

There is no requirement that all, or even most, packets belong to flows, i.e., carry non-zero flow labels. This observation is placed here to remind protocol designers and implementors not to assume otherwise. For example, it would be unwise to design a router whose performance would be adequate only if most packets belonged to flows, or to design a header compression scheme that only worked on packets that belonged to flows.

## 7. Priority

The 4-bit Priority field in the IPv6 header enables a source to identify the desired delivery priority of its packets, relative to other packets from the same source. The Priority values are divided into two ranges: Values 0 through 7 are used to specify the priority of traffic for which the source is providing congestion control, i.e., traffic that "backs off" in response to congestion, such as TCP traffic. Values 8 through 15 are used to specify the priority of traffic that does not back off in response to congestion, e.g., "real-time" packets being sent at a constant rate.

For congestion-controlled traffic, the following Priority values are recommended for particular application categories:

```
0 - uncharacterized traffic
1 - "filler" traffic (e.g., netnews)
2 - unattended data transfer (e.g., email)
3 - (reserved)
4 - attended bulk transfer (e.g., FTP, NFS)
5 - (reserved)
6 - interactive traffic (e.g., telnet, X)
7 - internet control traffic (e.g., routing protocols, SNMP)
```

For non-congestion-controlled traffic, the lowest Priority value (8) should be used for those packets that the sender is most willing to have discarded under conditions of congestion (e.g., high-fidelity video traffic), and the highest value (15) should be used for those packets that the sender is least willing to have discarded (e.g., low-fidelity audio traffic). There is no relative ordering implied between the congestion-controlled priorities and the non-congestion-controlled priorities.

# 8. Upper-Layer Protocol Issues

## 8.1 Upper-Layer Checksums

Any transport or other upper-layer protocol that includes the addresses from the IP header in its checksum computation must be modified for use over IPv6, to include the 128-bit IPv6 addresses instead of 32-bit IPv4 addresses. In particular, the following illustration shows the TCP and UDP "pseudo-header" for IPv6:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Payload Length                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      zero                     |  Next Header  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- If the packet contains a Routing header, the Destination Address used in the pseudo-header is that of the final destination. At the originating node, that address will be in the last element of the Routing header; at the recipient(s), that address will be in the Destination Address field of the IPv6 header.

- The Next Header value in the pseudo-header identifies the upper-layer protocol (e.g., 6 for TCP, or 17 for UDP). It will differ from the Next Header value in the IPv6 header if there are extension headers between the IPv6 header and the upper-layer header.

- The Payload Length used in the pseudo-header is the length of the upper-layer packet, including the upper-layer header. It will be less than the Payload Length in the IPv6 header (or in

the Jumbo Payload option) if there are extension headers between the IPv6 header and the upper-layer header.

- o Unlike IPv4, when UDP packets are originated by an IPv6 node, the UDP checksum is not optional. That is, whenever originating a UDP packet, an IPv6 node must compute a UDP checksum over the packet and the pseudo-header, and, if that computation yields a result of zero, it must be changed to hex FFFF for placement in the UDP header. IPv6 receivers must discard UDP packets containing a zero checksum, and should log the error.

The IPv6 version of ICMP [RFC-1885] includes the above pseudo-header in its checksum computation; this is a change from the IPv4 version of ICMP, which does not include a pseudo-header in its checksum. The reason for the change is to protect ICMP from misdelivery or corruption of those fields of the IPv6 header on which it depends, which, unlike IPv4, are not covered by an internet-layer checksum. The Next Header field in the pseudo-header for ICMP contains the value 58, which identifies the IPv6 version of ICMP.

## 8.2 Maximum Packet Lifetime

Unlike IPv4, IPv6 nodes are not required to enforce maximum packet lifetime. That is the reason the IPv4 "Time to Live" field was renamed "Hop Limit" in IPv6. In practice, very few, if any, IPv4 implementations conform to the requirement that they limit packet lifetime, so this is not a change in practice. Any upper-layer protocol that relies on the internet layer (whether IPv4 or IPv6) to limit packet lifetime ought to be upgraded to provide its own mechanisms for detecting and discarding obsolete packets.

## 8.3 Maximum Upper-Layer Payload Size

When computing the maximum payload size available for upper-layer data, an upper-layer protocol must take into account the larger size of the IPv6 header relative to the IPv4 header. For example, in IPv4, TCP’s MSS option is computed as the maximum packet size (a default value or a value learned through Path MTU Discovery) minus 40 octets (20 octets for the minimum-length IPv4 header and 20 octets for the minimum-length TCP header). When using TCP over IPv6, the MSS must be computed as the maximum packet size minus 60 octets, because the minimum-length IPv6 header (i.e., an IPv6 header with no extension headers) is 20 octets longer than a minimum-length IPv4 header.

## Appendix A. Formatting Guidelines for Options

This appendix gives some advice on how to lay out the fields when designing new options to be used in the Hop-by-Hop Options header or the Destination Options header, as described in section 4.2. These guidelines are based on the following assumptions:

- o One desirable feature is that any multi-octet fields within the Option Data area of an option be aligned on their natural boundaries, i.e., fields of width n octets should be placed at an integer multiple of n octets from the start of the Hop-by-Hop or Destination Options header, for n = 1, 2, 4, or 8.

- o Another desirable feature is that the Hop-by-Hop or Destination Options header take up as little space as possible, subject to the requirement that the header be an integer multiple of 8 octets long.

- o It may be assumed that, when either of the option-bearing headers are present, they carry a very small number of options, usually only one.

These assumptions suggest the following approach to laying out the fields of an option: order the fields from smallest to largest, with no interior padding, then derive the alignment requirement for the entire option based on the alignment requirement of the largest field (up to a maximum alignment of 8 octets). This approach is illustrated in the following examples:

Example 1

If an option X required two data fields, one of length 8 octets and one of length 4 octets, it would be laid out as follows:

```
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                | Option Type=X |Opt Data Len=12|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         4-octet field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                         8-octet field                         +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Its alignment requirement is 8n+2, to ensure that the 8-octet field starts at a multiple-of-8 offset from the start of the enclosing

header. A complete Hop-by-Hop or Destination Options header containing this one option would look as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  | Hdr Ext Len=1 | Option Type=X |Opt Data Len=12|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         4-octet field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                         8-octet field                         +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Example 2

If an option Y required three data fields, one of length 4 octets, one of length 2 octets, and one of length 1 octet, it would be laid out as follows:

```
                                                +-+-+-+-+-+-+-+-+
                                                | Option Type=Y |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Opt Data Len=7 | 1-octet field |         2-octet field         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         4-octet field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Its alignment requirement is 4n+3, to ensure that the 4-octet field starts at a multiple-of-4 offset from the start of the enclosing header. A complete Hop-by-Hop or Destination Options header containing this one option would look as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  | Hdr Ext Len=1 | Pad1 Option=0 | Option Type=Y |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Opt Data Len=7 | 1-octet field |         2-octet field         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         4-octet field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| PadN Option=1 |Opt Data Len=2 |       0       |       0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Example 3

A Hop-by-Hop or Destination Options header containing both options X and Y from Examples 1 and 2 would have one of the two following formats, depending on which option appeared first:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  | Hdr Ext Len=3 | Option Type=X |Opt Data Len=12|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         4-octet field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                         8-octet field                         +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| PadN Option=1 |Opt Data Len=1 |       0       | Option Type=Y |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Opt Data Len=7 | 1-octet field |         2-octet field         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         4-octet field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| PadN Option=1 |Opt Data Len=2 |       0       |       0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  | Hdr Ext Len=3 | Pad1 Option=0 | Option Type=Y |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Opt Data Len=7 | 1-octet field |         2-octet field         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         4-octet field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| PadN Option=1 |Opt Data Len=4 |       0       |       0       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       0       |       0       | Option Type=X |Opt Data Len=12|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         4-octet field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                         8-octet field                         +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

## Security Considerations

This document specifies that the IP Authentication Header [RFC-1826] and the IP Encapsulating Security Payload [RFC-1827] be used with IPv6, in conformance with the Security Architecture for the Internet Protocol [RFC-1825].

## Acknowledgments

The authors gratefully acknowledge the many helpful suggestions of the members of the IPng working group, the End-to-End Protocols research group, and the Internet Community At Large.

## Authors' Addresses

Stephen E. Deering
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
USA

Phone: +1 415 812 4839
Fax: +1 415 812 4471
EMail: deering@parc.xerox.com

Robert M. Hinden
Ipsilon Networks, Inc.
2191 E. Bayshore Road, Suite 100
Palo Alto, CA 94303
USA

Phone: +1 415 846 4604
Fax: +1 415 855 1414
EMail: hinden@ipsilon.com

## References


[RFC-1825] Atkinson, R., "Security Architecture for the Internet Protocol", RFC 1825, Naval Research Laboratory, August 1995.

[RFC-1826] Atkinson, R., "IP Authentication Header", RFC 1826, Naval Research Laboratory, August 1995.

[RFC-1827] Atkinson, R., "IP Encapsulating Security Protocol (ESP)", RFC 1827, Naval Research Laboratory, August 1995.

[RFC-1885] Conta, A., and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 1885, Digital Equipment Corporation, Xerox PARC, December 1995.

[RFC-1884] Hinden, R., and S. Deering, Editors, "IP Version 6 Addressing Architecture", RFC 1884, Ipsilon Networks, Xerox PARC, December 1995.

[RFC-1191] Mogul, J., and S. Deering, "Path MTU Discovery", RFC 1191, DECWRL, Stanford University, November 1990.

[RFC-791] Postel, J., "Internet Protocol", STD 5, RFC 791, USC/Information Sciences Institute, September 1981.

[RFC-1700] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/Information Sciences Institute, October 1994.

[RFC-1661] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, Daydreamer, July 1994.